A Security API for Distributed Social Networks

Michael Backes
Saarland University and
MPI-SWS
Germany
backes@mpi-sws.org

Matteo Maffei
Saarland University
Germany
maffei@cs.uni-saarland.de

Kim Pecina
Saarland University
Germany
pecina@cs.uni-saarland.de

Abstract

We present a cryptographic framework to achieve access control, privacy of social relations, secrecy of resources, and anonymity of users in social networks. We illustrate our technique on a core API for social networking, which includes methods for establishing social relations and for sharing resources. The cryptographic protocols implementing these methods use pseudonyms to hide user identities, signatures on these pseudonyms to establish social relations, and zero-knowledge proofs of knowledge of such signatures to demonstrate the existence of social relations without sacrificing user anonymity. As we do not put any constraints on the underlying social network, our framework is generally applicable and, in particular, constitutes an ideal plug-in for decentralized social networks.

We analyzed the security of our protocols by developing formal definitions of the aforementioned security properties and by verifying them using ProVerif, an automated theorem prover for cryptographic protocols. Finally, we built a prototypical implementation and conducted an experimental evaluation to demonstrate the efficiency and the scalability of our framework.

1. Introduction

Over the last years, online social networks (OSNs) have become the natural means to get in touch with people and to engage in a number of social activities, such as sharing information, exchanging opinions, organizing events, and publishing advertisements. The new dimensions of social interaction and the opportunities deriving from these novel functionalities tend to push into the background the impressive leakage of personal information (e.g., religious beliefs, political opinions, and sexual orientations) and the consequent threats to users' privacy.

It is well-understood that the lack of *access control mechanisms* (e.g., to restrict the access to pictures, videos, and posts on the wall) may lead to unpleasant consequences, such as employers monitoring the personal life of their employees. Nowadays, most centralized social networks (e.g., Facebook [49]) implement their own access control mechanisms (we refer to a recent study by Carminati and Ferrari [17] for a comprehensive overview of access control mechanisms in social networks) and, recently, cryptographic solutions [51, 22, 45, 8] have been proposed to enforce access control schemes in distributed social networks.

The threats to the privacy of users, however, go well beyond access control problems. For instance, OSNs are highly vulnerable to coercion attacks, where the coercer asks the user for the password, accesses its profile, and learns the list of friends together with their recent activities. This kind of attacks have been reported in countries ruled by authoritarian governments [27, 38, 1, 54], where people use social networks to organize protest activities and to publish documents that would be censored otherwise. In such settings, the *privacy of social relations* is a fundamental property as well as *user anonymity*. Combining anonymity with forms of access control is crucial to ensure that certain documents can only be read or posted by friends as opposed to hostile users. Anonymity and privacy properties are also desirable in many other applications that can run of top of social networks, such as content sharing and feedback reports.

In this paper, we present a cryptographic framework to achieve a wide range of security properties in OSNs, including access control, privacy of social relations, secrecy of resources, and anonymity of users. We illustrate our technique on a core API for social networking, which includes methods for establishing social relations and for sharing resources. Our framework does not constrain the underlying social network and, in particular, constitutes an ideal plug-in for distributed social networks [34, 47, 50, 8, 45], where social information and OSN functionality are decoupled and users can choose where and how to store their data.[1] Dis-

[1] We remark that our API could be implemented on top of centralized social networks as well, for example, to bypass the native access control

tributed social networks allow users to strengthen their control over sensitive material and, in addition, facilitate the management and interoperability of different networks and services.

Our contributions. The contribution of this paper is threefold:

- *Cryptographic Framework.* We present a number of cryptographic protocols to securely implement the methods of our API. In a nutshell, the fundamental idea is to use pseudonyms to hide user identities, signatures on these pseudonyms to establish social relations, and non-interactive zero-knowledge proofs of knowledge[2] of such signatures to demonstrate the existence of social relations. More precisely, a principal A establishes a social relation with principal B by signing B's pseudonym and a tag $\mathcal{R}$ describing the social relation. The resulting signatures are sent to B in encrypted form and are kept secret by the two principals. B can demonstrate to be in a certain social relation $\mathcal{R}$ with A by proving the knowledge of a signature on its pseudonym or the knowledge of a signature on the tag $\mathcal{R}$, depending on whether B wants to reveal its identity to A or not.

  Access control lists are defined in terms of pseudonyms (which by themselves do not reveal user identities) and social relations. Besides the resources themselves, access control lists constitute the only information stored on the servers. Signatures are only needed by the prover while anonymously authenticating with the verifier and can be stored on some secure device while offline. Hence, an attacker compromising the server of a principal cannot get any information about its friends, not even after reading the access control lists and monitoring the incoming authentication requests.

- *Performance evaluation.* We developed a prototypical implementation of our cryptographic framework and conducted an experimental evaluation to demonstrate its efficiency. The zero-knowledge proofs, which dominate the communication and computational complexity of our protocols, are a few kilobytes in size and their generation and verification takes less than one second on average.

- *Formal security analysis.* We provide a formal and automated security proof of our protocols. We specify the cryptographic protocols in a process calculus [3], we formalize access control policies and secrecy requirements as trace properties, and we characterize the anonymity guarantees provided by our protocols in terms of observational equivalence relations. We consider a strong adversarial model where the attacker has the control over the social relations, the access control policies, some of the protocol parties, and the protocols executed by the users.

  The security properties are successfully verified using ProVerif [12], a state-of-the-art automated theorem prover based on Horn clause resolution that provides security proofs for an unbounded number of protocol sessions and parties.

Related Work. Privacy and anonymity in online social networks are an emerging security problem, as witnessed by the vast literature on this topic [9, 23, 39, 42, 51, 46, 8, 45]. Weaknesses of and attacks on OSNs account for a large part of that literature. To the best of our knowledge, this work presents the first generally applicable framework to formally define and to provably achieve anonymity of users, secrecy of exchanged resources, and privacy of social graphs in OSNs.

Our work is close in spirit to some recent OSN projects [34, 47, 22, 50, 8, 45].

Safebook [22, 23] by Cutillo, Molva, and Strufe is a distributed social network implementation based on a special peer-to-peer overlay network, named matryoshka after its shape. The idea of Safebook is to capitalize on the trust relations among users to achieve integrity and privacy properties. Intuitively, external observers do not learn any information on the social graph as messages are passed from the outside of the matryoshka to its inside. In our approach, the signatures that witness the friendship relations are stored on a secure external medium and thus not only external observers but even attackers that successfully compromise the server of a user do not learn any information on the social graph. The access control in Safebook is based on public-key cryptography and users have no means to hide their identities from the other parties involved in the protocol. We provide a more fine-grained approach by assigning each resource an individual access control list and by harnessing the power of zero-knowledge proofs to provide authentication modalities that do not sacrifice user anonymity. Finally, our approach does not rely on any particular network topology.

Persona [8] by Baden et al. implements a distributed social network with distributed data storage. Access control is obtained by employing a combination of traditional public-key cryptography and attribute-based encryption (ABE)

mechanisms and achieve more fine-grained security properties.

[2] A zero-knowledge proof combines two seemingly contradictory properties. First, it is a proof of a statement that cannot be forged, i.e., it is impossible, or at least computationally infeasible, to produce a zero-knowledge proof of a wrong statement. Second, a zero-knowledge proof does not reveal any information besides the bare fact that the statement is valid [31]. A non-interactive zero-knowledge proof is a zero-knowledge protocol consisting of one message sent by the prover to the verifier. A zero-knowledge proof of knowledge additionally ensures that the prover knows the witnesses to the given statement.

scheme. In ABE schemes, attributes such as "neighbor" are assigned to users and data is encrypted such that only participants holding the assigned attributes can decrypt the data. The combination of classical public-key schemes and ABE schemes has the drawback of increased key management complexity. Our implementation uses only traditional public-key cryptography and, consequently, only two key pairs need to be stored. Also, our approach does not require a trusted third party for the key setup as necessary in ABE schemes.

The social network scheme proposed by Sun, Zhu, and Fang [45] is closely related to Persona, however, access control along with an efficient revocation mechanism is achieved by using broadcast encryption [28]. This social network scheme also requires a trusted third party for key management. Due to the distributed storage on untrusted network sites, a considerable computation effort is required to enable searches on encrypted data. In our approach, data servers are operated and maintained by the data owner and search queries can be performed on unencrypted data without computational overhead.

Both of these worksmainly specify the cryptographic implementation of their corresponding APIs, the security proofs are not formal and do not consider possible unintended interleavings of API calls. Sun et al. discuss how their scheme can be modified to allow for an anonymous social network but their model significantly differs from ours: they are mainly concerned with maintaining the anonymity of the resource providers. We on the other hand guarantee the unconditional anonymity of the requester. Further, we have formalized all desired security properties such as authenticity, secrecy, and anonymity, and proven them using ProVerif, an automated theorem prover that provides security proofs for an unbounded number of protocol sessions. In particular, this excludes all unintended protocol interleavings. In addition, our approach does not require any trusted third parties.

Mezzour et al. solve the problem of finding a relationship path in social networks where the relationships are kept secret [37]: the idea is to use cryptographic token propagation and a private set intersection protocol together with homomorphic encryption, which allows for path discovery even when users are offline. Our work focuses also on the enforcement of anonymity properties and we thus describe a protocol that allows us to anonymously register a "friend of a friend" relation rather than an algorithm for path discovery. Due to the nature of our protocols and the degree of freedom regarding the employed encryption scheme, however, we might use the technique Mezzour et al. to obtain a sophisticated way of finding social acquaintances in a private social graph.

Lockr [51] by Tootoonchian et al. is a cryptographic framework to achieve access control and privacy of social graphs. Lockr is among the first architectures to propose a separation of the social graph from the content of social networks and it shows that this is possible even without the support of the underlying OSN. Nonetheless, the authentication protocol in Lockr does not preserve the anonymity of users. The attacker can thus retrieve the social relations of a principal by simply monitoring the incoming authentication requests. Our approach allows users to authenticate using pseudonyms or social relations (thus hiding their identity), and we even offer the option to authenticate anonymously without revealing anything other than the knowledge of a signature.

Although the setting is different, our work may resemble the anonymous delegatable credential scheme [10] by Camenisch et al. which enables a root authority to issue anonymous credentials that can further be delegated. Our protocol intentionally does not allow for such a delegation. Further, the credential scheme is explicitly rooted at a certain node, i.e., a credential can always be assigned to the original root authority. In our scheme, any user can act as a source and it is hence sufficient to only issue a signature once and for all rather than once for every possible origin.

Backes et al. have recently introduced the concept of anonymous webs of trust [4]. The goal of that work is to allow principals to prove trust relations in an anonymous way, which is achieved by deploying zero-knowledge proofs. Although these proofs can be used to prove complex trust relations, they are in general expensive and turn out to be practical only for short key sizes.

We finally mention that neither group signatures [19, 11, 16] nor ring signatures [43, 53] suffice to provide the properties we achieve by means of zero-knowledge proofs. Group signatures usually rely on a group manager that sets up the group and can reveal the creator of a particular signature (traceability). Consequently, it is impossible to provide the anonymity of the requester as a corrupt group manager is able to immediately determine the originator of a request. Notice that the revocation can also be used to break our targeted anonymity property by consecutively revoking the signing capabilities of all members in a particular group. Further, each group membership requires a separate key, increasing the key management complexity. Ring signatures cannot be used either as the public keys of all ring members need to be publicly known, which reveals the social graph.

Outline of the paper. Section 2 introduces our API. Section 3 gives an overview of the cryptographic protocols and Section 4 describes the cryptographic setup in detail. Section 5 illustrates the performance evaluation and the experimental results. Section 6 gives a formal and automated security proof of our protocols. Section 7 concludes and gives directions for further research.

$\mathcal{M}$	::=	masks		M	::=	register (B,p_B)
	| p	pseudonym			|	getHandles ($\mathcal{M}_B$)
	| $\mathcal{R}$	social relation			|	getResource ($\mathcal{M}_B$,$hdl(res)$)
op	::=	operations			|	putResource ($\mathcal{M}_B$,$hdl(res)$, res')
	$r \mid w \mid rw$				|	getFriends ($\mathcal{M}_B$)
ACL	::=	access control list			|	indirectRegister ($\mathcal{M}_B$,C)
	($\mathcal{M}$,op)::ACL | [ ]					

We let A, B, and C range over principals. We write $hdl(res)$ to denote the handle of the resource res.

Table 1. Grammar of access control lists

2. A Core API for Social Networking

This section describes a security API for social networking, which includes methods to establish social relationships as well as to upload and download resources. In this paper, we do not aim at specifying a fully fledged API, since this would comprise a variety of application-dependent methods that are similar to each other and present the same problem (e.g., getImage, getVideo, readPost, etc.). We rather focus on a concise set of methods, which suffice to encode the others (e.g., a single primitive getResource is sufficient to encode the aforementioned methods) and allow us to deal with all the problems related to the cryptographic realization of an API for social networking.

A central feature of our API is that social links are kept secret and principals can engage in social activities (e.g., post a comment or retrieve a picture) without disclosing their identities.

We could not keep social relations private if access control lists revealed the identity of the principals with read and write capabilities: an attacker compromising the server of principal A would be able to read the access control lists stored therein and immediately learn the identity of A's friends. For this reason, access control lists are defined on *masks* (cf. Table 1), which are ranged over by $\mathcal{M}$ and consist of either a *pseudonym* p or a *social relation* $\mathcal{R}$. The idea is that a user B communicates its pseudonym p_B while establishing social relations: B is the only user that can use this pseudonym and only the users whom B registered with know the link between p_B and B (the pseudonym itself does not reveal any information about the owner's identity, cf. Section 3.2). We do not impose constraints on the usage of pseudonyms: B can decide to always use the same pseudonym such that friends can track all its activities or to use different pseudonyms to become unlinkable. The social relation $\mathcal{R}$ is simply a tag characterizing a certain social relation. An access control list consists of a list of pairs, whose first component is a mask and the second component is an operation (e.g., r, w, and rw). For instance, $[(p_B, rw), (\textit{friends}, r)]$ is an access control list specifying that the associated resource can be read and written by the principal with pseudonym p_B and it can additionally be read by the principals in the *friends* relation.

The protocols comprising our API are designed to protect the social relations and the anonymity of principals against external observers and against attackers that compromise the servers running the API. A dishonest principal can of course reveal its social relations but an attacker that observes the network traffic or breaks into an API server should not be able to learn them. The key idea to achieve this strong anonymity property is that principals can get and post resources by simply revealing their pseudonym or by proving to be in a certain social relation with the resource provider. Our cryptographic realization does not require the resource provider to store any information about its social relations, just the access control lists. This means that an attacker compromising the server of the resource provider cannot get any information about its friends, not even after reading the access control lists and monitoring the incoming authentication requests. The requester, instead, has to prove to be associated to a certain pseudonym or to be in a certain social relation with the resource provider. This procedure requires the knowledge of some information about the social relations that is, however, only needed when a principal goes online and wants to authenticate. Hence, these data do not need to be stored on a server (they can be stored, for example, on a secure portable device) and they are not leaked in case of server compromise.

Note that an ACL does *not* reveal enough structure about the social graph to apply de-anonymization techniques [39]. An access control list typically consists of social relations that do not reveal any structural information on the social graph. Should a user decide to mainly use pseudonyms, "padding" the ACL with fake pseudonyms (e.g., stipulating that social relatives register only fresh pseudonyms and adding fake pseudonyms until all resources have 1000 pseudonyms associated with them) suffices to hide the actual structure of the social graph and to render de-anonymization techniques inapplicable. Such a blinding technique has no consequence for the requester and only a negligible computational overhead for the resource provider. Moreover, since all signatures, i.e., the so-

cial relations, and the pseudonym-user bindings are stored on a well-hidden external device, even complete access to a number of servers will not reveal any social relation.

We now describe the methods composing the API, which are summarized in Table 1. We write $A.\mathsf{M}$ to denote the method M exported by A's API.

$\mathcal{R} \leftarrow A.\mathsf{register}(B, p_B)$**:** This method takes as input the identifier B of the principal that wants to establish a social relationship with A and a pseudonym p_B chosen by B (cf. Section 3.2). This method returns a tag $\mathcal{R}$ chosen by A to characterize the social relation (e.g., a string such as "friends" or a number). The cryptographic realization ensures that the caller corresponds to the identifier specified in the argument. The pseudonym p_B can be used by A to allow user-based access to B when setting up its access control list. Although A is able to link p_B to B's identity, the pseudonym itself does not reveal anything about B's identity, which is crucial to achieve anonymity even if A's server is compromised, the access control list is leaked, and the incoming authentication requests are monitored.

$hdl(res_1), \ldots, hdl(res_n) \leftarrow A.\mathsf{getHandles}(\mathcal{M}_B)$**:** This method takes as input the caller's mask $\mathcal{M}_B$ and returns the handles to A's resources. A handle identifies and describes the resource without disclosing it.[3] Handles are passed to the other methods to specify the resource of interest, as discussed below. Since this method takes as input the caller's mask, B has the option to reveal its pseudonym or to stay anonymous by just proving to be in a certain social relation with A, depending on whether A accepts anonymous requests or reveals its handles only to non-anonymous requesters.

$res \leftarrow A.\mathsf{getResource}(\mathcal{M}_B, hdl(res))$**:** This method takes as input the caller's mask $\mathcal{M}_B$ and the handle $hdl(res)$ of the requested resource. If $\mathcal{M}_B$ is given read access to res in the corresponding access control list, $\mathsf{getResource}$ returns the resource res.

$ack \leftarrow A.\mathsf{putResource}(\mathcal{M}_B, hdl(res), res')$**:** This method takes as input the caller's mask $\mathcal{M}_B$, the handle $hdl(res)$ of the resource to modify, and the new content res'. This method encodes the methods used by B to post comments on A's wall, to upload pictures, and so on. Typically, messages and pictures are appended while other resources such as the profile picture are replaced. We intentionally leave the actual behavior of this method unspecified, since it depends on the specific social network and on the kind of resource.

$\mathcal{R} \leftarrow A.\mathsf{indirectRegister}(\mathcal{R}_{AB}, p_B, C)$**:** This method allows users to establish indirect social relations (e.g., "friend of a friend"). It takes as input the social relation $\mathcal{R}_{AB}$ between A and B, a pseudonym p_B chosen by B and the identifier C of the principal which B is interested in establishing an indirect relation with. Notice that B must be in a direct social relation with A, and A has to be in a direct social relation with C. The idea is that if C accepts indirect relations, B can ask A to establish an indirect relation with C on his behalf. This method returns a tag $\mathcal{R}$ that describes the newly established social relation between C and B.

This method could in principle be cascaded to establish indirect relationships of arbitrary degree (e.g., "friend of a friend of a friend" relations). Since such relations are not used in practice, we do not consider them further here.

$C_1, \ldots, C_n \leftarrow A.\mathsf{getFriends}(\mathcal{M}_B)$**:** This method takes as input the caller's mask $\mathcal{M}_B$ and returns the list of A's friends that accept indirect relations. Notice that B must be in a direct social relation with A and the identities of A's friends that do not accept indirect social relations are not disclosed.

The API additionally comprises standard functions to deal with access control lists (e.g., creation and modification). Since these operations are local, they do not need any cryptographic infrastructure and, for the sake of readability, we omit them throughout this paper.

3. Overview of the Cryptographic Protocols

We now describe the cryptographic protocols implementing the API described in Section 2.

3.1. Preliminaries

In the following, we let $\mathsf{sign}(M, \mathsf{sk}_B)$ denote the signature on message M with B's signing key sk_B, $\mathsf{check}(S, M, \mathsf{vk}_B)$ denote the verification of the signature S on message M with B's verification key vk_B (the verification succeeds if and only if $S = \mathsf{sign}(M, \mathsf{sk}_B)$), $\mathsf{enc}(M, \mathsf{ek}_B)$ denote the encryption of M with key ek_B, and $\mathsf{dec}(E, \mathsf{dk}_B)$ denote the decryption of E with B's decryption key dk_B (the decryption succeeds and returns M if and only if $E = \mathsf{enc}(M, \mathsf{ek}_B)$).

Borrowing the notation introduced by Backes et al. [6], we write $\mathsf{ZK}[\mathsf{stm};\ M_1, \ldots, M_m;\ N_1, \ldots, N_n]$ to symbolically represent a non-interactive zero-knowledge proof

[3] For the sake of generality, we do not specify the format of handles: for instance, one can use thumbnails as handles for pictures and URI-style descriptions for text documents.

of knowledge of the statement stm with private values $M_1, \ldots, M_m$ (these values are also called witnesses) and public values $N_1, \ldots, N_n$. The private values are kept secret by the proof while the public ones are revealed. The statement is built on place-holders α_i for the i-th private value and β_j for the j-th public values. In a way of example, $\mathsf{ZK}[\mathsf{check}(\alpha_1, \alpha_2, \beta_1);\ \mathsf{sign}(M, \mathsf{sk}_B), M;\ \mathsf{vk}_B]$ is a proof that the prover knows a signature made by B. Notice that the zero-knowledge proof reveals neither the signature nor the signed message; just the verification key is revealed. $\mathsf{ZK}[\mathsf{check}(\alpha_1, \beta_1, \beta_2);\ \mathsf{sign}(M, \mathsf{sk}_B);\ M, \mathsf{vk}_B]$ is similar but the message M is revealed to the verifier. We finally write $\mathsf{ver}(ZK, \mathsf{stm})$ to denote the verification of the zero-knowledge proof ZK (the verification succeeds if ZK is a zero-knowledge proof of statement stm and this statement holds true after replacing the place-holders with the corresponding private and public values). As we will see, the zero-knowledge proofs used in our cryptographic realization enjoy the *non-malleability* property, i.e., both secret and public values can be seen as hard-coded into the proof in such a way that they cannot be changed without recomputing the proof. The cryptographic implementation is described in detail in Section 4.

3.2. Pseudonyms

Pseudonyms are used to authenticate with friends, who know the binding between pseudonyms and principals. A basic property of pseudonyms is to be uniquely binding, meaning that only their owners should be able to use them for authentication purposes. This can be easily achieved in our framework by employing a technique inspired by the Pseudo-Trust protocol [36]: given a one-way function f, a principal can generate a random number r and let its pseudonym be $f(r)$. To avoid impersonation attacks, it suffices to send in the registration and authentication protocols a zero-knowledge proof of knowledge of the preimage of $f(r)$, which is hard to compute for any other principal given the one-way function property.

3.3. Revocation

Equipping pseudonym systems such as Pseudo Trust [36] and our API with a revocation mechanism is an open problem. In Pseudo Trust and in our API, a zero-knowledge proof shows possession of a credential. As this credential is hidden by the proof, credential-based revocation is hard to achieve. Typical revocation mechanisms such as revocation signatures also fail since one would have to prove that no revocation signature exists (this is a false statement, since the revocation signature exists even if it has not been computed yet).

Nonetheless, we can circumvent credential revocation while still obtaining similar results. In the spirit of anonymous webs of trust [4], we can periodically re-issue all credentials based on a global interval. The authenticating zero-knowledge proof would then additionally prove that the signature is valid, i.e., it belongs in the current interval.

One might also draw inspiration from Sun et al. [45] and use a broadcast encryption scheme to mimic revocation: when answering a get-request, the resource is encrypted using the broadcast encryption; a revoked receiver would not be able to decrypt the resource. Before accepting a put-request, a challenge-response protocol is executed to ensure that the key of the requester has not been revoked yet.

3.4. Cryptographic Protocols

We now give a detailed description of the cryptographic protocols implementing the API methods.

$\mathcal{R} \leftarrow A.\mathsf{register}(B, p_B)$**:** The protocol is depicted in Figure 1. B starts the registration procedure by encrypting two messages for A: a signature and a non-interactive zero-knowledge proof of knowledge. The signature is on A's identifier, a pseudonym p_B, and a fresh (symmetric) session key k to be used in the response. The zero-knowledge proof shows that the pseudonym is of the form $f(r)$ and that the prover knows the preimage r, thus ensuring that principals can only register with their own pseudonyms. Notice that the zero-knowledge proof does not reveal r, which is crucial to avoid impersonation attacks. We attach the intended receiver's identifier A to the proof, to ensure that A cannot reuse this proof to impersonate B with other users.[4] A replies by encrypting a signature on p_B and a signature on the tag $\mathcal{R}$ describing the social relation.[5] While the encryptions prevent potential eavesdroppers from learning the identity of the parties, the signatures used in this protocol ensure the integrity of the registration request and the registration response. The signatures in the response also constitute the basis of our zero-knowledge proofs. We remark that social relations are unidirectional but they can straightforwardly be made bidirectional by running twice the registration protocol.

$hdl(res_1), \ldots, hdl(res_n) \leftarrow A.\mathsf{getHandles}(\mathcal{M}_B)$**:** The protocol is depicted in Figure 2. B sends a zero-knowledge proof in encrypted form to authenticate

[4] Recall that our zero-knowledge proofs are non-malleable, i.e., the messages attached to the proof cannot be changed without re-computation of the proof, which requires knowledge of the secret witnesses.

[5] The presence of two distinct signatures as opposed to one signature on the concatenation of the two messages makes the generation and verification of the zero-knowledge proofs employed in the other protocols more efficient: if we were to sign the pseudonym and the tag together, we should split this pair in zero-knowledge in the other protocols, which would involve a higher number of range proofs.

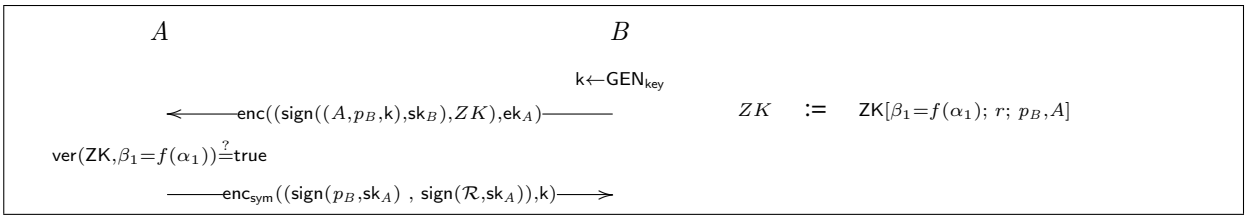

Figure 1. Protocol for register

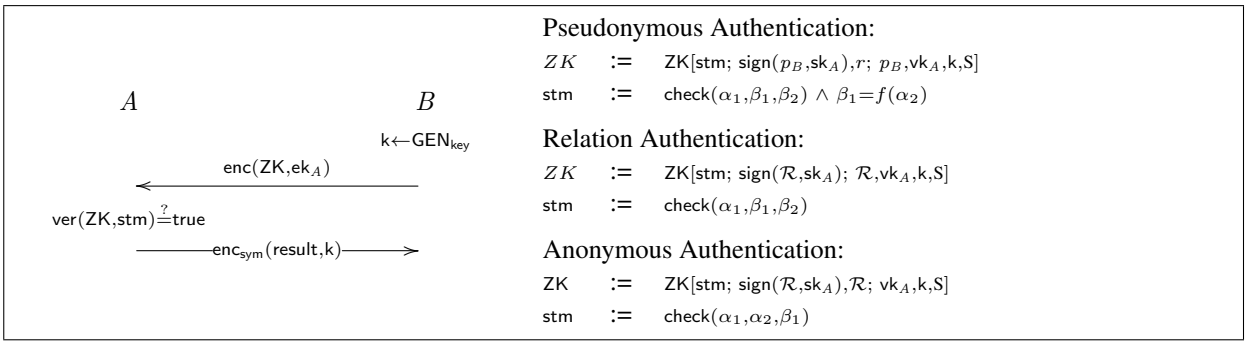

Figure 2. Protocol for getHandles**,**getResource**,**putResource**,** getFriends

with A. We provide three authentication modalities, namely, *pseudonymous authentication*, *relation authentication*, and *anonymous authentication*. In the pseudonymous authentication, B proves the knowledge of a signature from A on the pseudonym p_B, which is revealed by the proof, as well as the knowledge of the preimage of p_B. In the relation authentication, B proves the knowledge of a signature from A on the tag $\mathcal{R}$, which is revealed by the proof. In the anonymous authentication, B proves the knowledge of a signature from A, without revealing the signed message. A fresh session key k to be used in the response is attached to the zero-knowledge proof. A verifies the proof, checks which resource handles the prover has the permission to read, and sends them to the prover encrypted with the session key received in the first message. This protocol is used to implement also the getResource, putResource, getFriends methods. The only difference is that the additional arguments (e.g., $hdl(res)$ in the case of getResource) are attached to the proof (denoted as S in the picture) and the message encrypted in the response is in general the result of the method call (denoted as result in the picture).

The three authentication modalities give different anonymity guarantees and their usage depends on the required service (or resource) and on the access control list. For instance, if a certain resource res is protected by the access control list $[(p_B, rw), (\textit{friends}, r)]$, then B can run the relation authentication protocol to read res but B has to run the pseudonymous authentication protocol, thus revealing its identity to A, to write on res. In general, there is a trade-off between the restrictiveness of access control lists and the anonymity of requesters.

$\mathcal{R}_{CB} \leftarrow A$.indirectRegister$(\mathcal{R}_{AB}, p_B, C)$**:** The protocol is depicted in Figure 3. B sends an encrypted zero-knowledge proof to C, proving to be the owner of the pseudonym p_B. A session key k to be used in the final response from C is attached to this proof. B also sends an encrypted zero-knowledge proof to A, proving to be in a certain social relation $\mathcal{R}_{AB}$ with A. The pseudonym p_B is also attached to this proof. A verifies the proof and sends to C another zero-knowledge proof, showing to be in the social relation $\mathcal{R}_{CA}$ with C and stating to be in the social relation $\mathcal{R}_{AB}$ with the owner of p_B ($\mathcal{R}_{AB}$ and p_B are attached to the proof). C verifies the proof and sends in encrypted form to B a signature on the pseudonym p_B and a signature on a tag $\mathcal{R}_{CA}$ describing an indirect social relation obtained from the relations $\mathcal{R}_{CA}$ and $\mathcal{R}_{AB}$.

In order to allow for fine-grained access control policies, we additionally enable a similar protocol in which A authenticates with C by revealing its pseudonym p_A instead of the social relation $\mathcal{R}_{CA}$. This variant allows C to build more precise access control lists, built on relations of the form "friends-of-p_A" instead of "friends-of-friends".

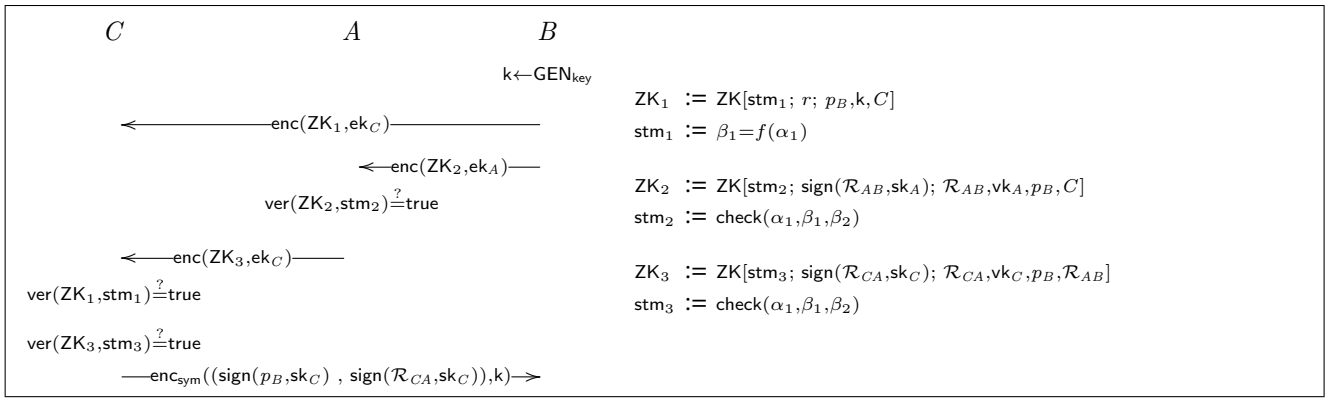

Figure 3. Protocol for indirectRegister

Concerning the anonymity guarantees provided by this protocol, A does not learn the identity of B, and C learns neither the identity of A nor the one of B (at most, the pseudonym of A is revealed to C in the pseudonymous authentication variant). Finally, since the session key is sent directly by B to C, A can neither read nor modify the signatures sent by C to A.

3.5. Discussion

Assumptions. Our protocols assume an established public-key infrastructure (PKI), i.e., all public keys are bound to their owner and this binding is verifiable. This can be realized by classical PKIs (e.g., VeriSign [52]), webs of trust (e.g. PGP [41]), or off-band communication (e.g., key signing parties [35]). Further, we assume an anonymous channel, i.e., a channel from which protocol participants can anonymously read and onto which they can anonymously write. In practice, such a channel can be realized using onion routing techniques [32, 26] or mix nets [18].

Attacker model and privacy properties. The attacker model considered in this paper comprises external observers eavesdropping the communication channel as well as attackers compromising the servers (e.g., by malware or coercion), reading the data stored thereon, and monitoring the subsequent incoming requests. The goal of our API is to preserve the privacy of social relations and to guarantee the anonymity of parties. Intuitively, external observers do not learn any information since all communication is encrypted. The only data revealing information about the social relations are the signatures released in the register and indirectRegister protocols. These signatures do not have to be stored on the servers, since they are proven in zero-knowledge, the verifier does not need them to verify the proof, and the prover only needs them while requesting a specific resource (they can be stored, for example, on a portable device and be hidden or destroyed in case of an attack). If a coercer asks the resource provider for the link between pseudonyms and principal identities, the resource provider can cheat and give fake associations since these cannot be verified by the coercer. Therefore, an attacker taking the control over the server of a principal does not gain any information about the social relations of that principal. In addition, a principal A can authenticate by revealing either its pseudonym, in which case A implicitly reveals its identity to the verifier who knows the link between pseudonyms and identities, or its social relation, in which case A obtains a form of k-anonymity in that its identity is hidden behind the number of people in the same social relation. We conducted a formal security analysis that is described Section 6.

Zero-knowledge proofs as anonymous signatures. We remark that the verification of the zero-knowledge proofs deployed in our protocols does not involve any secret information. In fact, these proofs can be verified by everybody, not only the intended verifier, and can thus serve as a kind of publicly verifiable anonymous signatures. In a way of example, the zero-knowledge proofs involved in the putResource protocol can be attached to the resource (e.g., a comment on A's wall) to provide a publicly verifiable information on the origin of that resource (e.g., a "friend of A").

4. Cryptographic Implementation

We now introduce the basic building blocks of our protocols and describe how they can be combined to achieve the desired zero-knowledge property. In the following we use the notation $e \in_R S$ to denote that e is drawn uniformly at random from the set S.

Camenisch-Lysyanskaya signature. This signature scheme was introduced Camenisch and Lysyanskaya [14] together with some zero-knowledge proofs. A public key is

a tuple $\mathsf{vk} = (a, b, c, n)$ where $n = p \cdot q$ is a special RSA modulus, i.e., $p = 2 \cdot p' + 1$, $q = 2 \cdot q' + 1$, and p, p', q, q' are primes. The numbers a, b, and c are uniformly random elements of $\mathsf{QR}(n)$, the group of quadratic residues modulo n. The corresponding secret key is $\mathsf{sk} = p$. Since factorizing n is assumed to be hard, the attacker cannot efficiently compute sk.

To sign a given message $m \in [0, \ldots, 2^{\ell_m})$, one chooses a random prime e of length $\ell_e \geq \ell_m + 2$ and a random number $s \in_{\mathsf{R}} [0, \ldots, 2^{\ell_m + \ell_n + \ell})$ where ℓ_m is an upper bound of the length of the messages to be signed, ℓ_n is the bit-length of n, and ℓ is a security parameter. In practice, $\ell = 160$ is considered secure. Finally, one computes v such that:

$$v \equiv_n (a^m \cdot b^s \cdot c)^{1/e} \tag{1}$$

Here and throughout this paper, we write $v \equiv_n u$ to say that u is equivalent to v modulo n. If the modulus is clear from the context, we often write $v = u$ to denote $v \equiv_n u$. Notice that the factorization of n is used to efficiently compute $1/e$. The signature on message m is the tuple $\mathsf{sig}_m = (e, s, v)$.

Given $\mathsf{vk} = (a, b, c, n)$, m, and $\mathsf{sig}_m = (e, s, v)$, the verification of the signature sig_m is performed by checking that $2^{\ell_e - 1} < e < 2^{\ell_e}$ along with the following equivalence:

$$v^e \equiv_n (a^m \cdot b^s \cdot c) \tag{2}$$

This equation constitutes the cryptographic instantiation of the predicate $\mathsf{check}(\mathsf{sig}_m, m, \mathsf{vk})$ discussed in Section 2. Under the commonly used strong RSA assumption, the Camenisch-Lysyanskaya signature scheme is secure against existential forgery attacks. Security against existential forgery is the standard notion of security when dealing with signature schemes.

Definition 1 (Strong RSA Assumption)**.** *Upon input of an RSA modulus n and an element $u \in \mathbb{Z}_n^*$, it is hard to compute values $e > 1$ and v such that $v^e \equiv u \mod n$. More formally, for all polynomial-time circuit families $\{\mathcal{A}_k\}$, there exists a negligible function $\mu(k)$ such that*

$$\mathsf{Pr}[n \leftarrow \mathsf{RSAmodulus}(1^k); u \leftarrow QR_n; \\ (v, e) \leftarrow \mathcal{A}_k(n, u) : e > 1 \wedge v^e \equiv_n u] = \mu(k)$$

Zero-knowledge proofs. Zero-knowledge proofs were first introduced by Goldwasser, Micali, and Rackoff [33] and have since then become a key element of many cryptographic protocols. A zero-knowledge proof is an interactive proof system (P, V) between two parties: the prover P and the verifier V. Both parties obtain the statement to be proven as input, the prover additionally receives a witness to the given statement. Besides the usual completeness and soundness properties, the zero-knowledge property ensures that even a malicious verifier cannot learn any information on the prover's witness.[6] Our zero-knowledge scheme builds on a very efficient class of zero-knowledge protocols, called Σ-protocols [21]. We briefly review below the basic building blocks of our scheme.

Σ-protocols and their properties. Σ-protocols comprise three message exchanges: *commitment* (com), *challenge* (ch), and *response* ($resp$), sent by the prover, the verifier, and the prover respectively. The protocols below enjoy the *special soundness* and the *special honest verifier statistical zero-knowledge (SHVSZK)* properties [21].

Special soundness is a strong form of *proof of knowledge* and guarantees that a prover is in possession of a witness. This property says that, given two protocol transcripts with the same commitment but different challenges, one can extract a witness to the proven statement. Honest verifier zero-knowledge is a variant of the zero-knowledge property where the verifier chooses the challenge uniformly at random from the challenge space and, in particular, independently of the commitment sent by the prover.[7] We write $\{\mathsf{ZK}(\widetilde{\alpha}) : S\}$ to denote a proof of knowledge of witnesses $\widetilde{\alpha}$ for statement S.

Σ-protocols can be combined together to prove logical conjunctions and disjunctions of their respective statements [21].

Commitments. A commitment scheme consists of the commit phase and the open phase. Intuitively, it is not possible to look inside a commitment until it is opened (hiding property) and the committing principal cannot change the content while opening (binding property). Using commitments, we hide the witnesses of a proof and, at the same time, we are able to perform computations on them. We use the integer commitment scheme by Damgård and Fujisaki [30, 25].

A commitment key comprises a special RSA modulus n of length ℓ_n, $g \in_{\mathsf{R}} \mathsf{QR}(n)$, and $h \in_{\mathsf{R}} \langle g \rangle$ where $\langle g \rangle$ denotes the group generated by g. To commit to an integer x, one draws $r \in_{\mathsf{R}} \mathbb{Z}_n$ and computes the commitment $C \equiv_n g^x h^r$. To open a commitment, one sends the tuple (x, r) to the verifier who checks whether $C \stackrel{?}{=} g^x h^r$ or not. We let $[\![C]\!]$ denote the value committed to in C.

Representation proofs. These proofs are used to show knowledge of a representation of an element y with respect to base elements $(g_1, \ldots, g_m)$. In particular, we use

[6] The zero-knowledge property is formalized using a simulator that, without having access to the witness to a given statement, creates simulated proof transcripts that are indistinguishable from actual protocol transcripts. Intuitively, this guarantees that the proof cannot be used to gain any information on the witness.

[7] In general, zero-knowledge implies honest-verifier zero-knowledge but the converse does not necessarily hold. In our setting, however, focusing on honest verifiers does not restrict the power of the attacker since the proof will eventually be made non-interactive using the Fiat-Shamir heuristic [29], which lets the prover herself choose the challenge by using the random oracle, without interacting with the verifier.

these proofs to show ownership of a pseudonym, i.e., to prevent impersonation attacks on pseudonyms. We write $\{\mathsf{ZK}((\alpha_i)_{i=1,\ldots,m}) : y = \prod_{i=1}^{m} g_i^{\alpha_i}\}$ to denote such representation proofs. If we use the base elements from the commitment scheme, we often write $\{\mathsf{ZK}(\alpha) : [\![C]\!] = \alpha\}$ to denote $\{\mathsf{ZK}(\alpha, \rho) : C = g^{\alpha} h^{\rho}\}$.

We use the representation proofs suggested by Fujisaki and Okamoto [30]. The prover chooses $r_1, ..., r_m \in_{\mathsf{R}} (0, 2^{\ell+t+\ell_m})$ and sends $w \equiv_n \prod_{i=1}^{m} g_i^{r_i}$ to the verifier. The prover responds to the challenge ch sent by the verifier with $s_i := r_i - ch \cdot x_i$, for $i = 1, \ldots, m$. The verifier accepts if and only if $w = y^{ch} \prod_{i=1}^{m} g_i^{s_i}$.

Proofs of equality of discrete logarithms. These proofs are used to show that two commitments y_1 and y_2 contain the same value (i.e., $[\![y_1]\!] = [\![y_2]\!]$). Such proofs can be used to "glue" together individual proofs. For instance, one can prove the knowledge of two discrete logarithms and additionally show that they coincide. We use the proof variant suggested by Camenisch and Lysyanskaya [14].

The prover chooses $r \in_{\mathsf{R}} (0, 2^{\ell+t+\ell_m})$ and $r_1, r_2 \in_{\mathsf{R}} (0, 2^{\ell+t+s} \cdot n)$, and computes the commitments $w_1 \equiv_n g_1^r h_1^{r_1}$ and $w_2 \equiv_n g_2^r h_2^{r_2}$. The prover responds to the challenge ch sent by the verifier with $s_c = r - ch \cdot x$ (in $\mathbb{Z}$), $s_1 = r_1 - ch \cdot x_1$ (in $\mathbb{Z}$), and $s_2 = r_2 - ch \cdot z_2$ (in $\mathbb{Z}$). The verifier accepts the proof if and only if $w_1 = y^{ch} \cdot g_1^{s_c} h_1^{s_1}$ and $w_2 = y_2^{ch} \cdot g_2^{s_c} h_2^{s_2}$.We write $\{\mathsf{ZK}(\alpha, \rho_1, \rho_2) : y_1 = g_1^{\alpha} h^{\rho_1} \wedge y_2 = g_2^{\alpha} h^{\rho_2}\}$ to denote proofs of the equality of discrete logarithms.

Range proofs. We use the range proofs proposed by Boudot [13]. A range proof guarantees that a certain committed value lies in the interval (A, B), where A and B are integers. Range proofs enable us to show the range requirement on the prime number e of the signature verification equation (2) in zero-knowledge. This proof will be denoted by $\{\mathsf{ZK}(\alpha) : [\![C]\!] = \alpha \wedge A < \alpha < B\}$. Notice that this proof does not reveal α, just the commitment C, and the bounds A and B of the interval. For technical reasons, these range proofs require an external commitment key (g_c, h_c, n_c) such that the discrete logarithm of h_c in basis g_c as well as the factorization of n_c are unknown. We hide this detail for the sake of readability.

Proving knowledge of a signature on a committed value. Given a commitment key (g, h, n), a commitment $D_x \equiv_n g^x h^{r_x}$ on a message x, commitments $D_e \equiv_n g^e h^{r_e}$, $D_s = g^s h^{r_s}$, and $D_v \equiv_n v g^w$ on the signature (e, s, v) of x, and a verification key $\mathsf{vk} = (a, b, c, n)$ that verifies the signature on x, the protocol in Figure 4 proves that (D_e, D_s, D_v) are commitments to a signature on the value x committed to in D_x, verifiable by vk with auxiliary commitments $D_w \equiv_n g^w h^{r_w}$. The two range proofs enforce the signature verification (2) requirement that the message x is at least two bits less in size than the prime number e. This protocol constitutes the cryptographic implementation of the proof $\mathsf{ZK}[\mathsf{check}(\alpha_1, \alpha_2, \beta_1);\ \mathsf{sign}(\mathsf{m}, \mathsf{sk}), \mathsf{m};\ \mathsf{vk}]$ from Section 3. $\mathsf{ZK}[\mathsf{check}(\alpha_1, \beta_1, \beta_2);\ \mathsf{sign}(\mathsf{m}, \mathsf{sk});\ \mathsf{m}, \mathsf{vk}]$, the proof in which message m is revealed, is obtained by opening the commitment to m. Camenisch and Lysyanskaya prove the following theorem as Lemma 15 [14].

Theorem 1. *The proof in Figure 4 constitutes a special honest verifier statistical zero-knowledge proof of knowledge (SHVSZK) with special soundness of a signature on message x.*

It remains to show how to prove pseudonym ownership, how to make a proof non-interactive, and how to attach messages to a proof in a non-malleable way.

Pseudonyms. As mentioned in Section 3.5, given a one-way function f, a principal can enforce the uniqueness of its pseudonym by generating a random number r and letting its pseudonym be $f(r)$. To avoid impersonation attacks, it suffices to send in the registration and authentication protocols a zero-knowledge proof of knowledge of the preimage of $f(r)$ to demonstrate ownership of the corresponding pseudonym.

We use modular exponentiation $g^r = p$ in a group $\langle g \rangle$, where computing discrete logarithms is intractable, and representation proofs to show the knowledge of preimages. Intuitively, only the owner of a pseudonym can create the corresponding representation proof as this requires the knowledge of the discrete logarithm r which is assumed to be hard to compute given only g and p. This proof constitutes the cryptographic realization of the proof $\mathsf{ZK}[\beta_1 = f(\alpha_1);\ r;\ p_B]$ from Section 3. Being a Σ-protocol, this proof can be combined with the proof of Figure 4 to prove the logical conjunction of the two statements (cf. the pseudonymous authentication protocol from Figure 2).

Non-interactive proofs and non-malleability. All the zero-knowledge proofs discussed so far are interactive protocols. The Fiat-Shamir heuristic [29] allows us to turn the aforementioned *interactive Σ-protocols* into *non-interactive zero-knowledge proofs of knowledge* in the random oracle model. Intuitively, a random oracle is a black-box that can be queried by protocol participants: each response to a query returns a uniformly random answer that is consistent with previous queries. Using the Fiat-Shamir heuristic, the statement and the commitment (i.e., the first message sent in a Σ-protocol) are hashed by the prover and the resulting value is used as a challenge. Since the prover can compute all these values herself, the proof consists of one single message sent by the prover to the verifier. This message consists of the commitment and the response (the verifier must recompute the challenge). In practice, we implement the random oracle as a cryptographic hash-function

$$\left\{ \begin{array}{l} \mathsf{ZK}(\chi, \sigma, \varphi, \omega, (\rho_i)_{i=1,2,3}) : \\ c = D_v^{\epsilon} \cdot (1/a)^{\chi} \cdot (1/b)^{\sigma} \cdot (1/g)^{\varphi} \ \wedge \ D_w = g^{\omega} h^{\rho_1} \ \wedge \ 1 = D_w^{\epsilon} \cdot (1/g)^{\varphi} \cdot (1/h)^{\rho_2} \ \wedge \ D_x = g^{\chi} h^{\rho_3} \ \wedge \\ 2^{\ell_e - 1} < \epsilon < 2^{\ell_e} \ \wedge \ 0 \leq \chi < 2^{\ell_m} \end{array} \right\}$$

Figure 4. Zero-knowledge proof of knowledge of a signature on value χ.

from the SHA or the MD family such as SHA-1 or MD6. We refer the interested reader to a study by Coron et al. [20] for a detailed discussion about the properties of these families.

The Fiat-Shamir heuristic can also be used to attach additional values v to a proof. The trick is to hash the attached values along with the statement and the commitment while computing the challenge. The resulting proof is non-malleable, as changing the attached values results in a different challenge (and hence a verification error).

Encryption schemes. Our cryptographic implementation is parametric in the deployed encryption scheme. Since the encryption scheme is merely used to protect the privacy of data and does not play any role in zero-knowledge proofs, we can in fact choose any off-the-shelf encryption scheme.

5. Experiments

Since Σ-protocols may be efficient or tremendously expensive, depending on which properties are proven and which values are kept secret,[8] we developed a prototypical implementation to test the computational and communication complexity of our cryptographic framework. We conducted the experimental evaluation on a standard notebook with a 2.5 GHz dual core processor and 4 GB memory. Our prototype is written in Java with NTL [44] accelerated with GMP [48] for the mathematical operations. We set the security parameters discussed in Section 4 to the following commonly used values: $\ell := 80$, $s := 160$, and $t := 80$. To determine the size of a proof, we serialized the corresponding Java object and used a compression algorithm to reduce the overhead caused by the Java serialization routine.

Proofs of knowledge of a signed relation tag. Figure 5 and Figure 6 show the generation time, the verification time, and the proof size of the zero-knowledge proofs of knowledge of a signed relation tag that are deployed in the protocol from Figure 2 (relation authentication[9] and anonymous authentication variants). In Figure 5, we fix the size of the signed messages (20 bytes, which typically suffice to describe social relations) and let the key size vary (in the interval 512-4096 bits). The time required to generate and verify these proofs is less than 1 second for key sizes up to 2560 bits. NIST classifies 2048-bit keys as secure until 2030 [40]. The size of these proofs is very small, just a few kilobytes. In Figure 6, we fix the key size (2048 bits) and let the size of the signed message vary (in the interval 16-128 bytes). The experimental results show that increasing the size of signed messages has an irrelevant impact on the efficiency of these proofs.

Proofs of pseudonym ownership. Figure 7 shows the generation time, the verification time, and the proof size of the zero-knowledge proofs of ownership of a pseudonym that are deployed in the protocols from Figure 1 and Figure 3. The experimental results show that these proofs do not have a noticeable impact on the overall performance of our protocols: generating and verifying these proofs takes a few milliseconds and their size is negligible.

Proofs of knowledge of a signed pseudonym. Figure 8 shows the generation time, the verification time, and the proof size of the proof of knowledge of a signed pseudonym that is deployed in the pseudonymous authentication variant of the protocol from Figure 2. Since both the key-size and the pseudonym-size are determined by the security parameter n, we let them vary together (in the interval 512-4096 bits). The experimental results are very similar to the ones in Figure 5, as expected given the similarities between these proofs.

Signatures and encryptions. Figure 9 shows that signatures have a modest impact on the overall performance of our protocols, which is largely determined by zero-knowledge proofs. We finally recall that our protocols do not put constraints on the encryption scheme and thus one can choose any efficient one.

Scalability. The computational and communication complexity of zero-knowledge proofs is essentially independent of the size of attached messages (i.e., those messages that do not serve as witnesses). The reason is that these messages are hashed together into the challenge of the zero-knowledge proof and hash-functions such as SHA-1 can hash several hundred megabytes of payload per second [24]. Such hashes can even be performed offline by storing the

[8] For instance, proving the mathematical exponentiation of two secret (i.e., committed) values is very expensive [15].

[9] The results for the proofs ZK_2 and ZK_3 from Figure 3 are the same as the ones for the proof used in the relation authentication variant of the protocol from Figure 2.

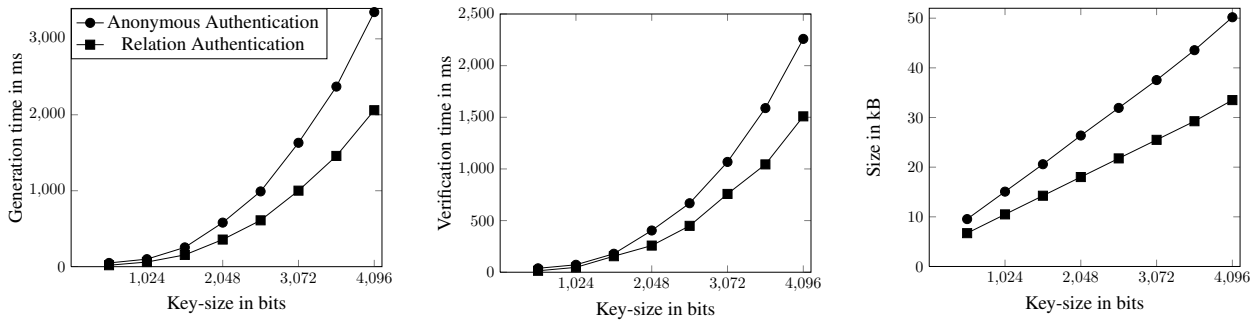

Figure 5. Proofs of knowledge of signed relation tags (fixed tag size, 20 bytes).

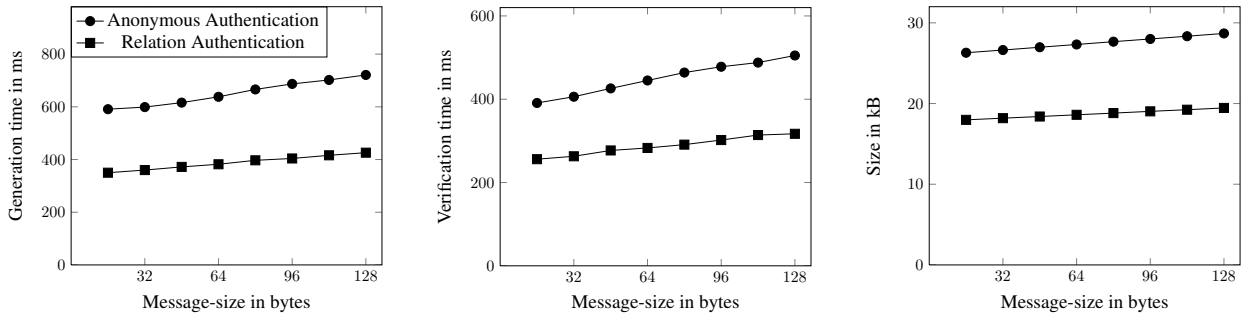

Figure 6. Proofs of knowledge of signed relation tags (fixed key size, 2048 bytes).

state of the hash-function together with the message itself. This means that our zero-knowledge proofs can be used even in settings where users want to transfer large-size data (e.g., movies).

Users typically store two signatures for each person they register with. As signatures small in size, managing one thousand friends requires only about 50 MB of storage, i.e., significantly less than the capacity of modern a USB stick.

We finally remark that the prime numbers that are needed to sign messages can be computed offline. Signing relation tags that average 50 bytes[10] requires 402-bit prime numbers, which can be sampled quickly. Signing a pseudonym computed with a 2048-bit prime number, instead, requires a 2050 bit prime number, whose sampling takes more time. The pseudonym size, however, is fixed and such large prime numbers can be precomputed offline.

The combination of efficient zero-knowledge proofs, fast hash-functions, small signature sizes, and the possibility to pre-compute primes enables our framework to easily scale to large size OSN.

6. Formal Verification

Although we have shown in Section 4 that the zero-knowledge proofs fulfill the desired statistical non-interactive zero-knowledge property, we have still to prove that the protocols as a whole do not suffer from attacks exploiting the protocol logic (e.g., unexpected interleavings between different protocol sessions, impersonation attacks, etc.) and provide the intended security properties. This is achieved by conducting a formal and automated security analysis. More precisely, we model our protocol in the applied pi-calculus [3], we formalize access control and secrecy as trace properties, we characterize the anonymity guarantees provided by our protocols in terms of observational equivalence relations, and we verify our model with ProVerif [12, 2], a state-of-the-art automated theorem prover that provides security proofs for an unbounded number of protocol sessions. We model zero-knowledge proofs following the approach by Backes et al. [6], for which computational soundness results exist [7]. The ProVerif scripts are available online [5].

6.1. Verification of Access Control and Secrecy Properties

We define access control and secrecy properties as trace properties and we verify that all the execution traces of the applied pi-calculus process $\mathcal{M}^{\mathsf{trace}}$, formalizing our model, fulfill these properties. The idea is to decorate the protocol with events (i.e., logical predicates) and to verify that in all execution traces these events are executed under some conditions and in a certain order. For instance, we want

[10] "friend of a friend" requires 18 bytes

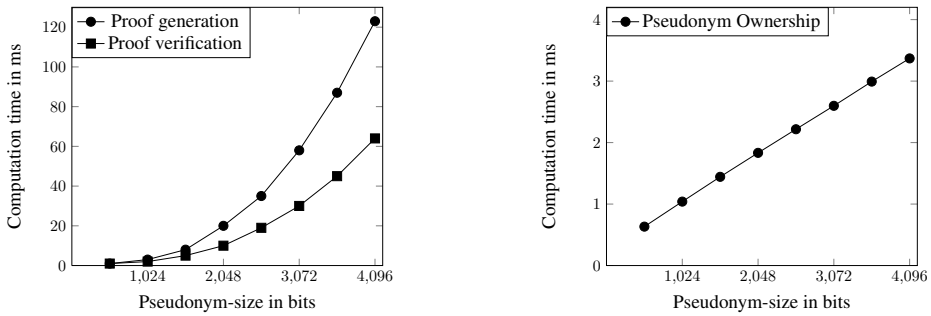

Figure 7. Proofs of pseudonym ownership.

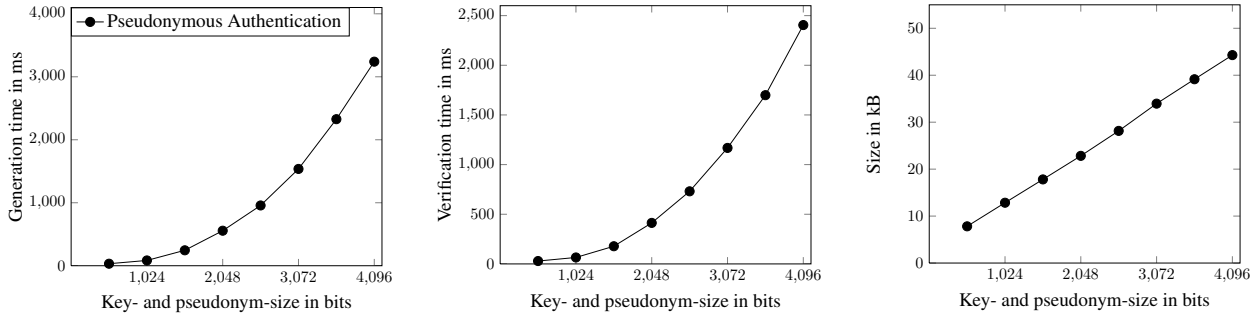

Figure 8. Proofs of knowledge of a signed pseudonym.

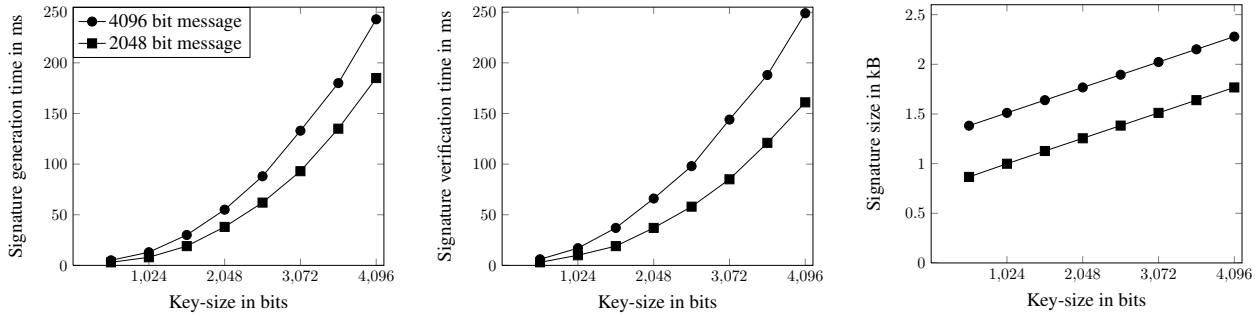

Figure 9. Signatures.

to check that whenever a principal sends a resource to a friend, then a friend must have requested that resource and an external observer does not learn that resource.

System description and attacker model. In our model we consider an unbounded number of users running the protocols of our API. For the sake of simplicity, we consider just one social relation, namely "friends", we partition the set of users into trusted and untrusted ones, and we assume that the resources of each trusted principal can be accessed only by its friends.

The attacker is a standard Dolev-Yao active adversary with full control over the public channels. The attacker may compromise and take the control over untrusted users. The attacker additionally dictates the protocols executed by trusted users and, in particular, rules their friendship relations. Since we are interested in protecting the resources against users that are not authorized to access them, we keep the invariant that untrusted users are never in the "friends" relation with trusted ones. Notice that this invariant is not a restriction and is actually necessary to make the access control properties meaningful: they would vacuously hold if also the attacker could legitimately access honest principals' resources.

Protocol annotations and trace properties. In the following, we concentrate on the register protocol from Figure 1, the getResource protocol from Figure 2, and the indirectRegister protocol from Figure 3. We first show the events used to decorate these protocols and then introduce the corresponding trace properties. We omit the events and the trace properties for other protocols from Figure 2, which

are, however, similar to the ones for the getResource protocol.

The annotations for the register protocol are shown in Figure 10. Before starting the protocol to register with A under the pseudonym p_B, B raises the event $\mathsf{BeginReg}(p_B, A, B)$. Before establishing a social relation $\mathcal{R}$ with B under the pseudonym p_B, A raises the event $\mathsf{Reg}(\mathcal{R}, p_B, A, B)$. Finally, after receiving the response from A, B raises the event $\mathsf{EndReg}(\mathcal{R}, A, B)$. The trace property is as follows

$$\begin{aligned}&\mathsf{EndReg}(\mathcal{R}, p_B, A, B) \Rightarrow \\ &\quad (\mathsf{Reg}(\mathcal{R}, p_B, A, B) \Rightarrow \mathsf{BeginReg}(p_B, A, B))\end{aligned} \tag{3}$$

meaning that all occurrences of events of the form $\mathsf{EndReg}(\mathcal{R}, p_B, A, B)$ have to be preceded by the event $\mathsf{Reg}(\mathcal{R}, p_B, A, B)$, which has in turn to be preceded by the event $\mathsf{BeginReg}(p_B, A, B)$.

The getResource protocol is annotated as shown in Figure 10. The only subtlety is that the binding between the handle h and the resource m is expressed via the event $\mathsf{isHandleOf}(h, m)$. Initially, this event is raised with $h = hdl(m)$. Whenever the putResource protocol is executed, the binding between h and the new resource m' is updated by raising the event $\mathsf{isHandleOf}(h, m')$.[11] The trace property for this protocol is as follows:

$$\begin{aligned}&\mathsf{RcvdRes}(\mathcal{R}, \mathsf{m}) \Rightarrow \\ &\quad ((\mathsf{ReleaseRes}(\mathcal{R}, \mathsf{m}) \Rightarrow \mathsf{RequestRes}(\mathcal{R}, h)) \,\&\, \\ &\quad \mathsf{isHandleOf}(h, \mathsf{m}))\end{aligned} \tag{4}$$

The annotations for the indirectRegister protocol are shown in Figure 11. The trace property for the indirectRegister protocol is as follows:

$$\begin{aligned}&\mathsf{EndReg2}(\mathcal{R}_{CB}, p_B, C, A, B) \Rightarrow \\ &\quad (\mathsf{Reg2}(\mathcal{R}_{CB}, \mathcal{R}_{CA}, \mathcal{R}_{AB}, p_B, C) \Rightarrow \\ &\quad\quad (\mathsf{Med}(\mathcal{R}_{CA}, \mathcal{R}_{AB}, p_B, C, A) \Rightarrow \\ &\quad\quad\quad \mathsf{BeginReg2}(\mathcal{R}_{AB}, p_B, A, B, C)))\end{aligned} \tag{5}$$

Even though this property might resemble property (3), there are some subtle, yet important, differences. Most notably, in the indirectRegister protocol neither C nor A knows who requested a relationship, which is reflected by B's missing identifier in the events Reg2 and Med.

We are finally interesting in verifying that the attacker cannot learn any of the resources shared by trusted principals (cf. below). Such resources are ranged over by m in the process $\mathcal{M}^{\mathsf{trace}}$. This secrecy property is expressed in ProVerif as:

$$\neg\, \mathsf{attacker}(\mathsf{m}) \tag{6}$$

[11] ProVerif does not support the removal of events; hence, the trace property captures the fact that released resource must have been bound at some point to the handle of interest, without necessarily being the resource currently bound to it. This is just an artifact due to the theorem prover used in the analysis.

Analysis results. The aforementioned trace properties have successfully been analyzed using ProVerif. This static analysis constitutes an automated proof of the following theorem.

Theorem 2 (Trace Properties)**.** *The trace properties formalized in equations* (3)*,* (4)*, and* (5) *and the secrecy property formalized in equation* (6) *hold true in all execution traces of the applied pi-calculus process* $\mathcal{M}^{\mathsf{trace}}$.

6.2. Verification of the Anonymity Property

Intuitively, we formalize the anonymity property as a cryptographic game where a distinguisher $\mathcal{D}$ has to tell whether the protocols of our API are executed by principal P_1 or principal P_2. If it is not possible to distinguish between these two scenarios, then the protocols guarantee anonymity. In ProVerif such an indistinguishability property is formalized as an *observational equivalence relation* $\mathcal{M}_1^{\mathsf{Anon}} \approx \mathcal{M}_2^{\mathsf{Anon}}$ between two processes $\mathcal{M}_1^{\mathsf{Anon}}$ and $\mathcal{M}_2^{\mathsf{Anon}}$. Figure 12 depicts the most important features of this observational equivalence relation. For the sake of presentation, we concentrate on the register protocol and the getResource protocol.

System description and attacker model. The indistinguishability game is set up as follows. The attacker is a standard Dolev-Yao active adversary with full control over the public channels. In both processes, P_1 and P_2 may register with both honest principals and compromised ones (i.e., controlled by the attacker). P_1 and P_2 can also run the getResource protocol with any of the principals they are registered with. We consider a strong attacker model in which the distinguisher dictates the protocols executed by all the users, including P_1 and P_2. Hence, the distinguisher controls the topology of the whole social graph.

Indistinguishability game for the register **protocol.** At some point, the distinguisher may ask the test principal (i.e., P_1 in the process $\mathcal{M}_1^{\mathsf{Anon}}$ and P_2 in the process $\mathcal{M}_2^{\mathsf{Anon}}$) to run the registration protocol with a certain trusted principal (cf. Figure 12). The attacker should not be able to tell which of the two processes is executed, i.e., who between P_1 and P_2 is running the registration protocol. In this case, the attacker is an external observer that can just look at the exchanged messages. Notice that the attacker cannot ask the test principal to register with a compromised user, since parties have to reveal their identity in the register protocol.

Indistinguishability game for the getResource **protocol.** At some point, the distinguisher may also ask the test principal to run the getResource protocol with a certain trusted principal or with a compromised one. In the former case, the distinguisher may ask the test principal to run any of the three authentication variants, while in the latter case, we

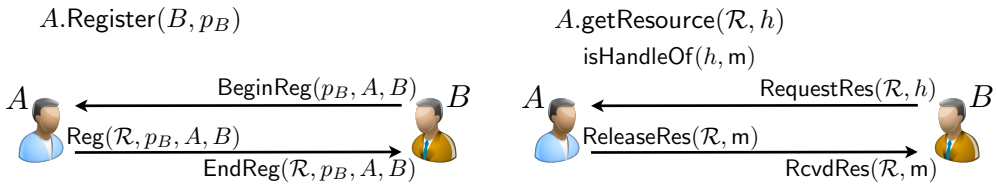

Figure 10. register **protocol and** getResource **protocols annotated with events.**

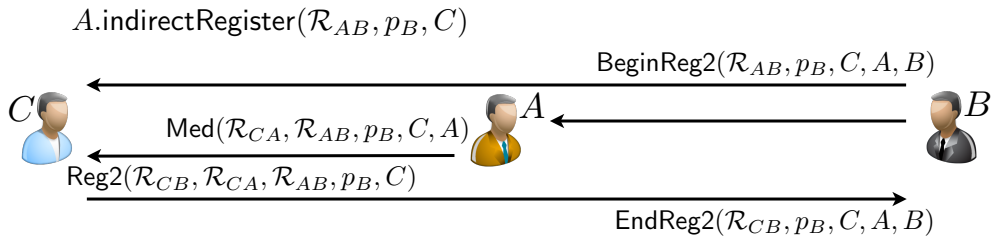

Figure 11. indirectRegister **protocol annotated with events.**

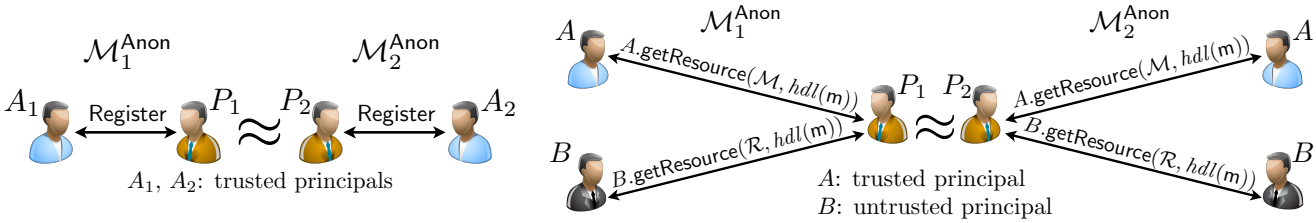

Figure 12. Observational equivalence relation for the getResource **protocol.**

disallow the pseudonymous authentication variant, as this variant is meant to reveal the identity of the requester. In the former case the distinguisher is a mere external observer, in the latter case it controls the resource provider. In both cases, A and B must be both registered with the resource provider.

Indistinguishability game for the other protocols. The indistinguishability game for the other protocols from Figure 2 is similar to the one for the getResource protocol, while the indistinguishability game for the indirectRegister protocol is similar to the one for the register protocol.

Analysis results. The observational equivalence relation described above has successfully been analyzed using ProVerif, which constitutes an automated proof of the following theorem.

Theorem 3 (Anonymity)**.** *For the two processes* $\mathcal{M}_1^{\mathsf{Anon}}$ *and* $\mathcal{M}_2^{\mathsf{Anon}}$*, the observational equivalence relation* $\mathcal{M}_1^{\mathsf{Anon}} \approx \mathcal{M}_2^{\mathsf{Anon}}$ *holds true.*

7. Conclusion

We presented a framework for achieving access control, privacy of social relations, secrecy of resources, and anonymity of users in social networks. Our framework relies on signatures to establish social relations and efficient zero-knowledge proofs to show the existence of such relations. Users can authenticate with each other by revealing their pseudonyms, their social relations, or by demonstrating to be in a certain social relation without revealing it. These authentication modalities provide different anonymity guarantees and allow for fine-grained access control policies. The usage of cryptography makes our framework an ideal plug-in for distributed social networks. The cryptographic protocols underlying our framework are very efficient and scale to large-size OSN.

We have developed a prototypical implementation of our API as a Facebook application [5]. This application allows users to protect their resources with expressive access control policies, to keep their social graph private, and to engage in anonymous communication. We are also looking at distributed social networks with the idea of developing a generally applicable plug-in.

Acknowledgments

This work was partially supported by the initiative for excellence and the Emmy Noether program of the German federal government and by the Miur Project SOFT (*Security Oriented Formal Techniques*).

References

[1] C. Abadi. Iran, facebook, and the limits of online activism. *Foreign Policy*, 2010. http://www.foreignpolicy.com/articles/2010/02/12/irans_failed_facebook_revolution.

[2] M. Abadi, B. Blanchet, and C. Fournet. Automated verification of selected equivalences for security protocols. In *Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 331–340. IEEE Computer Society Press, 2005.

[3] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL)*, pages 104–115. ACM Press, 2001.

[4] M. Backes, S. Lorenz, M. Maffei, and K. Pecina. Anonymous webs of trust. In *Proc. 10th Privacy Enhancing Technology Symposium (PETS)*, pages 130–148. Lecture Notes in Computer Science, 2010.

[5] M. Backes, M. Maffei, and K. Pecina. Source code and proverif models for trace properties and observational equivalences. http://lbs.cs.uni-saarland.de/sapi/.

[6] M. Backes, M. Maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Proc. 29th IEEE Symposium on Security & Privacy*, pages 202–215. IEEE Computer Society Press, 2008.

[7] M. Backes and D. Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *Proc. 21th IEEE Symposium on Computer Security Foundations (CSF)*, pages 255–269. IEEE Computer Society Press, 2008.

[8] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: An online social network with user-defined privacy. In *Proc. 2009 ACM SIGCOMM conference on Data communication*, pages 135–146. ACM Press, 2009.

[9] J. C. Baumgartner and J. S. Morris. Myfacetube politics: Social networking web sites and political engagement of young adults. *Social Science Computer Review*, 28:24–44, February 2010.

[10] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. In *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 108–125. Springer-Verlag, 2009.

[11] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *Topics in Cryptology - CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[12] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Press, 2001.

[13] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer-Verlag, 2000.

[14] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Proc. 3rd International Conference on Security in Communication Networks (SCN)*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer-Verlag, 2002.

[15] J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *Advances in Cryptology - EUROCRYPT 1998*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer-Verlag, 1998.

[16] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology - CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.

[17] B. Carminati and E. Ferrari. Privacy-aware access control in social networks: Issues and solutions. In *Privacy and Anonymity in Information Management Systems*, Advanced Information and Knowledge Processing, pages 181–195. Springer-Verlag, 2010.

[18] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1988.

[19] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology - EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.

[20] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-damgård revisited: How to construct a hash function. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer-Verlag, 2005.

[21] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.

[22] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: Secure social network. http://www.safebook.us/.

[23] L. A. Cutillo, R. Molva, and T. Strufe. Privacy preserving social networking through decentralization. In *Proc. 6th International Conference on Wireless On-Demand Network Systems and Services, WONS'09*, pages 133–140. IEEE Computer Society Press, 2009.

[24] W. Dai. Crypto++. http://www.cryptopp.com/benchmarks.html.

[25] I. Damgård and E. Fujisaki. An integer commitment scheme based on groups with hidden order. In *Proc. 8th International Conference on the Theory and Application of Cryptology and Information Security: ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer-Verlag, 2001.

[26] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, pages 303–320. USENIX Association, 2004.

[27] F. Fassihi. Iranian crackdown goes global. *The Wall Street Journal*, 2009. http://online.wsj.com/article/SB125978649644673331.html.

[28] A. Fiat and M. Naor. Broadcast encryption. In *Advances in Cryptology - CRYPTO 1993*, pages 480–491. Springer-Verlag, 1994.

[29] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO 1987*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.

[30] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology - CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, 1997.

[31] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991.

[32] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):39–41, 1999.

[33] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[34] D. Grippi, M. Salzberg, R. Sofaer, and I. Zhitromirskiy. DIASPORA*. http://www.joindiaspora.com.

[35] https://secure.wikimedia.org/wikipedia/en/wiki/Key_signing_party.

[36] L. Lu, J. Han, Y. Liu, L. Hu, J.-P. Huai, L. Ni, and J. Ma. Pseudo trust: Zero-knowledge authentication in anonymous p2ps. *IEEE Transactions on Parallel Distributed Systems*, 19(10):1325–1337, 2008.

[37] G. Mezzour, A. Perrig, V. Gligor, and P. Papadimitratos. Privacy-preserving relationship path discovery in social networks. In *Proc. 8th International Conference on Cryptology and Network Security*, pages 189–208. Springer-Verlag, 2009.

[38] E. Morozov. Foreign policy: Iran's terrifying Facebook police. *National Public Radio*, 2009. http://www.npr.org/templates/story/story.php?storyId=106535773.

[39] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *Proc. 30th IEEE Symposium on Security & Privacy*, pages 173 – 187. IEEE Computer Society Press, 2009.

[40] T. N. I. of Standards and Technology. Recommendataion for key management – part 1: General. *NIST Special Publications*, 800–57, 2007. http://csrc.nist.gov/groups/ST/toolkit/key_management.html.

[41] http://www.pgp.com/.

[42] V. Ponce, J. Wu, and X. Li. Improve peer cooperation using social networks. *International Journal on Parallel, Emergent and Distributed Systems*, 24(3):189–204, 2009.

[43] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Proc. 7th International Conference on the Theory and Application of Cryptology and Information Security: ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer-Verlag, 2001.

[44] V. Shoup. NTL: A library for doing number theory. http://www.shoup.net/ntl.

[45] J. Sun, X. Zhu, and Y. Fang. A privacy-preserving scheme for online social networks with efficient revocation. In *Proc. 29th conference on Information communications*, pages 2516–2524. IEEE Computer Society Press, 2010.

[46] G. Swamynathan, C. Wilson, B. Boe, K. Almeroth, and B. Y. Zhao. Do social networks improve e-commerce?: a study on social marketplaces. In *WOSP '08: Proceedings of the first workshop on Online social networks*, pages 1–6. ACM, 2008.

[47] T. A. Team. Appleseed Project. http://opensource.appleseedproject.org.

[48] T. G. Team. The GNU multiple precision arithmetic library. http://gmplib.org.

[49] http://www.facebook.com/.

[50] A. Thurston. DSNP: The distributed social network protocol. http://www.complang.org/dsnp.

[51] A. Tootoonchian, S. Sarouis, Y. Ganjali, and A. Wolman. Lockr: Better privacy for social networks. In *Proc. 5th International Conference on emerging Network Experiments and Technologies (CoNEXT)*, pages 169 – 180. ACM Press, 2009.

[52] http://www.verisign.com/.

[53] H. Wang and H. Yu. A novel signer-admission ring signature scheme from bilinear pairings. In *Proc. 1st International Workshop on Education Technology and Computer Science*, pages 631–634. IEEE Computer Society Press, 2009.

[54] D. Wolman. Cairo activists use Facebook to rattle regime. *WIRED Magazine*, 2008. http://www.wired.com/techbiz/startups/magazine/16-11/ff_facebookegypt.